Операторы и выражения

Операция присваивания

Операция присваивания выполняется справа налево и имеет две формы записи ***полную и короткую***.

Полная форма записи: *переменная = выражение;*
Примеры присваивания полной формы:

1. *Присваивание переменной константное значение***.**

```
int x;
x = 25;
```

2. *Присваивание переменной результат выражения*

```
double y;
y = (x+2) / (3.5*x) - 5;
```

3. *Операции присваивания, изменяющие значения нескольких операндов*

```
int x,y,z;
x = y = z = 5;
```

Сокращенная форма записи:

переменная <операция>= выражение;

где *операция* – арифметические или побитовые операции одна из операций (+ , -, *, /, %, &, |, ^, ~, >>,<<).
Примеры присваивания сокращенной формы:

x *= 5; ↔ x = x*5;
s += 7; ↔ s = s+7;
y /= x+3; ↔ y = y/(x+3);

Операция явного приведения типа

В любом выражении преобразование типов может быть осуществлено явно, для этого достаточно перед выражением поставить в скобках идентификатор соответствующего типа.

Вид записи операции*: (тип) выражение***;**

ее результат – значение выражения, преобразованное к заданному типу.
Операция приведения типа вынуждает компилятор выполнить указанное преобразование, но ответственность за последствия возлагается на программиста. Рекомендуется использовать эту операцию в исключительных случаях, например:
1. Для того чтобы правильно разделить два целых числа, необходимо при вычислении хотя бы один операнд явно преобразовать в тип double.

```
int x=5,y=2;
double z;
I.   z = (double)x/y;
II.  z = x/(double)y;
III. z = (double)x/(double)y;
printf("%d / %d = %.2lf\n",x,y,z);
```

2. При работе с функциями из библиотеке math.h

```
int x = 45;
sqrt((double)x);
```

Операции сравнения

== – равно или эквивалентно;
!= – не равно;
< – меньше;
<= – меньше либо равно;
> – больше;
>= – больше либо равно.

Общий вид операций отношений:

<выражение 1> <знак операции> <выражение 2>

результат операции отношения – значение 0, если отношение, ложно, в противном случае истинно. Следовательно, операция отношения может использоваться в любых арифметических выражениях.

Логические операции

Логические операции (в порядке убывания относительного приоритета) и их обозначения:

! – отрицание (логическое НЕТ);
&& – конъюнкция (логическое И);
|| – дизъюнкция (логическое ИЛИ).

Общий вид операции отрицания:

!<выражение>

Ненулевое значение операнда – истина, а нулевое – ложь, например:

!0 → 1
!5 → 0

Общий вид операций конъюнкции и дизъюнкции

<выражение 1> <операция> <выражение 2>

Например:
num1>10 && !(num2 % 7)→ истина, если 1-е и 2-е выражения истинны;
x>=0 || x==7 → истина, если хотя бы одно выражение истинно.

Особенность операций конъюнкции и дизъюнкции – экономное последовательное вычисление выражений-операндов:
<выражение 1> <операция> <выражение 2>
– если выражение 1 операции конъюнкция ложно, то результат операции ноль и выражение 2 не вычисляется;
– если выражение 1 операции дизъюнкция истинно, то результат операции единица и выражение 2 не вычисляется.

Операция «,» (запятая)

Данная операция используется при организации строго гарантированной последовательности вычисления выражений (используется там, где по синтаксису допустима только одна операция, а нам необходимо разместить две и более, например, в операторе for).

Форма записи: *выражение 1, …, выражение N;*
выражения 1,…,N вычисляются последовательно и результатом операции становится значение выражения N.

Например:

```
for (i = 0, j = n-1; i <= j; ++i, --j){ .....}
```

Операция sizeof

Операция sizeof позволяет определить размер типа данных или переменных (объектов) в оперативной памяти.

Операция sizeof имеет следующий вид: ***int sizeof****(имя)*
где имя – это *идентификатор* или *имя типа*.

Пример:

```
  short int num;
printf("short int=%d\n", sizeof(short int));
printf("short int = %d\n", sizeof (num));
```

Результат выполнения:
short int = 2
short int = 2

ОПЕРАТОРЫ УПРАВЛЕНИЯ ПРОЦЕССОМ ВЫПОЛНЕНИЯ АЛГОРИТМА

К управляющим инструкциям относятся:

1. операторы условного и безусловного переходов,
2. оператор выбора альтернатив (переключатель),
3. операторы организации циклов
4. операторы передачи управления (перехода).

Каждый из управляющих операторов имеет конкретную лексическую конструкцию, образуемую из ключевых слов языка Си, выражений и символов-разделителей.

Управляющие инструкции

Условный оператор

В языке Си имеется две разновидности условных операторов: *простой* и *полный*.

Синтаксис простого оператора:

if (*expression***)** *operator1;*

if (*expression***) {**
operator1;
operator2;
. . . . . .
operatorN;
}

В качестве ***выражения (*expression*)*** могут использоваться:

1. арифметическое или логическое выражение;
2. выражение сравнения;
3. целое число;
4. переменная целого типа;
5. вызов функции с соответствующим типом значения.

Если выражение истинно (не ноль), то выполняется оператор 1, иначе он игнорируется; оператор 1 – составной оператор или блок.

Примеры записи:

1) **if** (x > 0) x = 0;

2) **if** (i != 1) { j++; s = 1; } – последовательность операций;

3) **if** (! x) exit (1); ↔ **if** (x == 0) exit (1);

4) **if** (i > 0)

if (i < n) k++; ↔ **if** ((i > 0) && (i < n)) k++;

Синтаксис полного оператора условного выполнения:

if (*expression***)** *operator1;*
else *operator2;*

Если выражение не ноль (истина), то выполняется оператор 1, иначе – оператор 2; операторы 1 и 2 могут быть составным оператором или блоком.

Примеры записи:

```
if (x>0) j=k+10;
  else m=i+10;
```

Если есть вложенная последовательность операторов if-else, то else связывается с ближайшим предыдущим if, не содержащим else, например:

```
if (n > 0)
    if(a > b)  z = a;
    else   z = b;
```

Когда необходимо связать фразу else с внешним if, то используем операторные скобки:

```
if(n > 0)  {
    if (a > b)  z = a;
```

}
else z=b;

В следующей цепочке операторов if-else-if выражения просматриваются последовательно:

```
if (expression 1)  operator 1;
  else
    if (expression 2)  operator 2;
     else
       if (expression 3)  operator 3;
       else   operator 4;
```

Если какое-то выражение оказывается истинным, то выполняется относящийся к нему оператор и этим вся цепочка заканчивается. Последняя часть с else – случай, когда ни одно из проверяемых условий не выполняется. Когда при этом не нужно предпринимать никаких явных действий, else оператор 4; может быть опущен, или его можно использовать для контроля, чтобы засечь "невозможное" условие.

Пример:

```
if ( n < 0 ) printf ( "N отрицательное\n" );
     else
        if ( n==0 ) printf ( "N равно нулю\n );
       else printf ( "N положительное \n );
```

Тернарная операция " ? : ".

Формат написания условной операции:

***expression*1 ? *expression*2 : *expression*3;**

если выражение1 отлично от нуля (истинно), то результатом операции является выражение2, в противном случае – выражение3; каждый раз вычисляется только одно из выражений 2 или 3.

Для нахождения максимального значения из a и b (значение z) можно использовать оператор if :

```
if (a > b)  z=a;
else  z=b;
```

Используя условную операцию, этот пример можно записать как

```
z = ( a> b ) ? a : b;
```

Оператор выбора альтернатив

Общий формат оператора выбора альтернатив **switch**:

switch (*expression***)**{

```
  case const1:  operator1; break;
  case const2:  operator2; break;
  case const3:    operator3; break;
  case const4:
  case const5:    operator4; break;
  default:       operator5; break;
}
```

Значение вычисленного выражения должно быть целочисленного типа.

Оператор **break** выполняет выход из оператора **switch**. Если после оператора стоит **break**, то управление переходит к оператору, следующему за оператором **switch**. Если после оператора отсутствует оператор **break**, то выполняются все следующим операторы, пока не обнаружит оператора **break** или завершение оператора **switch**.

В случае несовпадения значения выражения с одной из констант происходит переход на **default** либо при отсутствии **default** – к оператору, следующему за оператором **switch**.

```
int i = 2;
enum number { ONE=1, TWO,  THREE };
switch(i) {
case ONE :     puts ( " CASE 1. "); break;
case TWO :     puts ( " CASE 2. "); break;
case THREE :  puts ( " CASE 3. "); break;
default:       puts ( " DEFAULT. "); break;
}
```